Eliminating Selectors from Term Rewriting Systems(Theory of Rewriting Systems and Its Applications)

AUTHOR(S):

Yutaka, Kikuchi

数理解析研究所講究録
918巻 1995年 268-276

Eliminating Selectors from Term Rewriting Systems

Kikuchi, Yutaka*
Tokyo Institute of Technology

95/07/26

Abstract

Function symbols of term rewriting systems (TRSs) can be classified into three categories, namely, constructors, selectors, and defined operators in terms of functional programming. The selectors are for convenience of programming and efficient execution of programs, so the constructors and the defined operators are sufficient to construct data structures and to manipulate them.

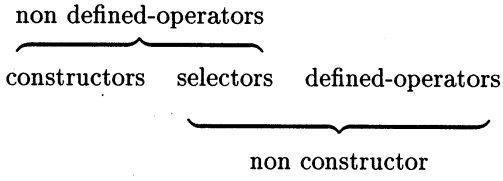
We give a transformation method to eliminate selectors from a TRS. We also introduce a new model of TRSs from the viewpoint of functional programming, and show that an original TRS and the transformed TRS have the almost same model except trivial differences.

The transformation method preserves strong termination property. Although the methods do not preserve weak terminating and confluency, we turn out sufficient conditions to keep those properties.

1 Introduction

We can divide function symbols into some categories when we consider TRSs as functional programs. Huet et al make a completion algorithm more efficient by dividing functions into two categories such that constructor and non-constructor[HH82].

We proceed to observe the role of functions in TRSs, non-constructor can be divide into further two categories, one is a set of selectors that access elements in a data structure constructed by constructors, the other is a set of defined operators that are the rest of functions.



Only constructors of a TRS derives any data structure on the TRS. Selectors corresponds to the constructors, which make programming convenient and make its execution efficient. Therefore there exists a TRS that has no selectors and has the same meaning intuitionally.

For example, assume a TRS whose constructors and rules are *cons*, *nil* and $car(cons(x,y)) \triangleright x$, $cdr(cons(x,y)) \triangleright y$, respectively. Either $2nd(x) \triangleright car(cdr(x))$ or $2nd(cons(x, cons(y,z))) \triangleright y$ works as a function that picks up the second element of a list.

It is interesting to transform a set of rules including selectors into that of rules that have no selectors. There are major two interests; one is how we can do that for any case of rules, the other is how to judge two TRSs have the same specification if the first interest is satisfied.

We will describe the concept of categorizing function symbols more precisely in the section 2. Next we will show a method to eliminate selectors from TRSs. In the section4 we will introduce a new model to

*E-mail: kikuchi@cs.titech.ac.jp

denote specifications of TRSs as functional programs. In the section 5, we will show some characteristic of the method. It turns out that the method does not preserve weak termination and confluency, so we will give some sufficient conditions of TRS to preserve the properties.

This paper is briefly translated from the technical paper [Kik95]. The transformationI is inspired from [BK86][GM87].

We use the same terminology of [DJ90] in this paper.

2 Constructors and Selectors

We regard a selector as a function that picks up an element from a data structure constructed by constructors. A constructor defines a set of selectors corresponding to the arity of the constructor therefore there is neither ambiguity nor voluntariness to define selectors. A set of rules is defined at the same time that specifies the relationship between the constructor and the selectors. There is no other rule that has a selector on the left hand side in the TRS.

Definition 2.1 *Let C is a set of constructors. A set of ground constructor terms $T(C)$ is a set of ground terms on C.*

Notation 2.2 *c_n is a constructor with arity $n(\geq 0)$. Constants are constructors whose arity are 0.*

Definition 2.3 *Corresponding to a constructor c_n, n-selectors $s_n^1, \ldots, s_n^n$ and n-rules $r_n^1, \ldots, r_n^n$ are defined as follows.*

$$r_n^i : s_n^i(c_n(x_1, \ldots, x_i, \ldots, x_n)) \triangleright x_i$$

This is the only rule in a TRS, which a selector s_n^i appears on the left hand side.

Notation 2.4 *We distinguish constructors by using a label l to express ${}_lc_n, {}_ls_n^i, {}_lr_n^i$ when there are different constructors that have the same arity.*

Notation 2.5 *s_n denotes one of $s_n^i (1 \leq i \leq n)$.*

Definition 2.6 *Defined functions are functions that are neither constructors nor selectors.*

Notation 2.7 *We use a notation $\langle R, \langle C, S, D\rangle, V\rangle$ to express explicitly a TRS $\langle R, F, V\rangle$ such that it has the constructors C, the selectors S and the defined operators D.*

The notation $\langle R, F, -\rangle$ or $\langle R, \langle C, S, D\rangle, -\rangle$ means that we do not care the variable set in the TRS.

We discuss a role of functions in a sense of functional programming and define a category of function symbols in this section according as the such role.

Huet et al showed that we could distinguish between constructors and selectors when a TRS was considered as a computation mechanism[HH82]. All constructor terms should be irreducible in the Huets' classification. They call those constructors free.

A condition that the constructors of a TRS are free makes the data structure derived from the TRS a strict well founded structure. The condition lets programming so inconvenient; for example, 0 and *succ*(*pred*(0)) are different values when we prepare the constructors 0, *succ*, *pred* to express integer numbers in a TRS. 0 and $-(0)$ are different values even if 0, *succ*, $-$ are the constructors.

We loose the condition so that we can define any constructors when we classify function symbols.

3 Eliminating selectors by rule transformations

We give a method to eliminate selectors from a given TRS by transforming rules of the TRS. We will discuss effects of the method in section5.

First phase is to use the following three transformation rules to remove each selector on the right hand side of a rule. The number of transforming operations is at most the number of selectors on the right hand side of the all rules in a TRS because each application of the transformations removes one or more selectors. Next phase is to remove all rules that contain selectors on the left hand side.

Trans I *Let $\langle R \cup \{p\}, \langle C, S, D\rangle, V\rangle$ be a TRS that the rule p contains selectors on the right hand side. We can express p as follows without loss of generality where γ is an arbitrary term.*

$$p\colon \gamma \triangleright \delta[\underbrace{s_n^{m_1}(\tau), \ldots, s_n^{m_k}(\tau)}_{s_n^{m_i}(\tau)}, \quad \underbrace{\tau, \ldots, \tau}_{\text{appearances of } \tau}\ ]$$

Target selectors are $s_n^{m_i} \in S$ that have τ as those arguments. We assume the number of the targets is k $(1 \leq i \leq k, 1 \leq m_i \leq n)$. The rest of the context δ is all appearances of τ with which s_n does not associate.

Remove the rule p and append two rules q_1, q_2 to remove the targets.

$$q_1\colon \gamma \triangleright h(\tau, \underbrace{y_1, \ldots, y_l}_{l-variables})$$

$$q_2\colon h(c_n(\underbrace{x_1, \ldots, x_n}_{x_i}), y_1, \ldots, y_l) \triangleright \delta[\underbrace{x_{m_1}, \ldots, x_{m_k}}_{x_{m_i}}, \underbrace{c_n(x_1, \ldots, x_n), \ldots, c_n(x_1, \ldots, x_n)}_{\text{appearances of } c_n(x_1, \ldots, x_n)}]$$

x_{m_i} and $c_n(x_1, \ldots, x_n)$ in q_1, q_2 replace $s_n^{m_i}(\tau)$ and τ without s_n association in p respectively. The transformed TRS is $\langle Q, F', V'\rangle$ where $P = R \cup \{p\}$, $Q = R \cup \{q_1, q_2\}$, $F = \langle C, S, D\rangle$, $D' = D \cup \{g\}$, $F' = \langle C, S, D'\rangle$, $V' = V \cup \{x_1, \ldots, x_n\}$.

h is a new defined operator called an auxiliary function symbol. $x_1, \ldots, x_n$ does not exist in P, $y_1, \ldots, y_l$ are the all appearances of variable in δ.

Trans II *Let $\langle R \cup \{p\}, \langle C, S, D\rangle, V\rangle$ be a TRS that the rule p is the following form.*

$$p\colon \gamma[\ \underbrace{x, \ldots, x}_{\text{appearances of } x}\ ] \triangleright \delta[\underbrace{s_n^{m_1}(x), \ldots, s_n^{m_k}(x)}_{s_n^{m_i}(x)}, \quad \underbrace{x, \ldots, x}_{\text{appearances of } x}\ ]$$

Target selectors are $s_n^{m^i}$ that have the same variable denoted by x. We assume the number of the targets is k $(1 \leq i \leq k, 1 \leq m_i \leq n)$.

The rest of the context δ is all appearances of x with which s_n does not associate.

Remove the rule p and append a rule q to remove the targets.

$$q\colon \gamma[c_n(x_1, \ldots, x_n), \ldots, c_n(x_1, \ldots, x_n)] \triangleright \delta[x_{m_1}, \ldots, x_{m_k}, c_n(x_1, \ldots, x_n), \ldots, c_n(x_1, \ldots, x_n)]$$

x_{m_i} and $c_n(x_1, \ldots, x_n)$ in q replace $s_n^{m_i}(x)$ and x without s_n association in p respectively. The transformed TRS is $\langle Q, F, V\rangle$ where $P = R \cup \{p\}$, $Q = R \cup \{q\}$, $F = \langle C, S, D\rangle$.

Trans III *Let $\langle R \cup \{p\}, \langle C, S, D\rangle, V\rangle$ be a TRS that the rule p is the following form.*

$$p\colon \gamma \triangleright \delta[s_n^i(c_n(\tau_1, \ldots, \tau_i, \ldots, \tau_n))]$$

A target selector is s_n^i. γ, $\tau_1, \ldots, \tau_n$ are arbitrary terms.

Remove the rule p and append a rule q to remove the target.

$$q\colon \gamma \triangleright \delta[\tau_i]$$

The transformed TRS is $\langle Q, F, V\rangle$ where $P = R \cup \{p\}$, $Q = R \cup \{q\}$, $F = \langle C, S, D\rangle$.

Trans IV *Let* $\langle R \cup \{p\}, \langle C, S, D\rangle, -\rangle$ *be a TRS that the rule* p *is the following form. Remove* p. *The transformed TRS is* $\langle R, F, -\rangle$ *where* $F = \langle C, S, D\rangle$

$$p{:}\,\gamma[s_n(\tau)] \triangleright \delta$$

3.1 Examples of transformation to eliminating selectors

We show some examples of selector elimination. Although we defined the notation of function symbols in the section2, conventional function names are used here to give much intuitionistic understanding.

Example 3.1 List manipulation
2nd picks up a second element of a list, *duplast* returns a list whose car and cdr are a last element of a given list and the whole given list respectively. *rev* returns a reverse order list of a given list, and the definition of *rev* is omitted to describe the TRS simply.

The functions should be defined as $nil = c_0$, $cons = c_2$, $car = s_2^1$, $cdr = s_2^2$ in accordance with the notation in the section2.

$$\left[\begin{array}{c} C = \{nil, cons\}, S = \{car, cdr\}, \\ D = \{2nd, rev, duplast\} \\ P = \left\{\begin{array}{c} car(cons(x,y)) \triangleright x \\ cdr(cons(x,y)) \triangleright y \\ 2nd(x) \triangleright car(cdr(x)) \\ duplast(x) \triangleright cons(car(rev(x)), x) \end{array}\right\} \end{array}\right]$$

The following is example reductions on the TRS.

$$2nd(cons(nil, cons(cons(nil, nil), nil))) \xrightarrow{!} cons(nil, nil)$$

$$duplast(cons(nil, cons(cons(nil, nil), nil))) \xrightarrow{!} cons(cons(nil, nil), cons(nil, cons(cons(nil, nil), nil)))$$

$$2nd(nil) \xrightarrow{!} car(cdr(nil))$$

$$duplast(nil) \xrightarrow{!} cons(car(nil), nil)$$

Note that there are normal ground terms that have selectors.

We get the following TRS from the previous TRS by the transformation method.

$$\left[\begin{array}{c} C = \{nil, cons\}, S = \emptyset, D' = \{2nd, rev, duplast, h\} \\ Q = \left\{\begin{array}{c} 2nd(cons(x_1, cons(y_1, y_2))) \triangleright y_1 \\ duplast(x) \triangleright h(rev(x), x) \\ h(cons(x_1, x_2), y_1) \triangleright cons(x_1, y_1) \end{array}\right\} \end{array}\right]$$

Any normal ground term on the transformed TRS has no selectors, while the original TRS not.

$$2nd(nil) \xrightarrow{!} 2nd(nil)$$

$$duplast(nil) \xrightarrow{!} h(nil, nil)$$

Example 3.2 *Natural number:*
– is a subtract function on natural numbers. The functions should be defined as $0 = c_0$, $succ = c_1$, $pred = s_1^1$ *in accordance with the section 2.*

$$\left[\begin{array}{c} C = \{0, succ\}, S = \{pred\}, D = \{-\} \\ P = \left\{\begin{array}{c} pred(succ(x)) \triangleright x \\ -(x, 0) \triangleright x \\ -(x, succ(y)) \triangleright -(pred(x), y) \end{array}\right\} \end{array}\right]$$

The following are examples on the TRS. There is a normal constructor term including a selector in the latter case.

$$-(succ(succ(0)), succ(0)) \xrightarrow{!} succ(0)$$

$$-(succ(0), succ(succ(0))) \xrightarrow{!} pred(0)$$

We get the following TRS by application of the transformation method.

$$\left[\begin{array}{c} C = \{0, succ\}, S = \emptyset, D = \{-\} \\ Q = \left\{\begin{array}{c} -(x, 0) \triangleright x \\ -(succ(x), succ(y)) \triangleright -(x, y) \end{array}\right\} \end{array}\right]$$

Now we have the following reduction sequence: $-(succ(0), succ(succ(0))) \xrightarrow{!} -(0, succ(0))$

4 Constructive Model

We propose new concepts about TRSs *constructive domain* and *constructive model.*

An initial model of a TRS is differ from our intuitionistic model when functions are categorized into three groups. For example in the introduction, we do not indicate that two rule sets describe the same specification because their initial models are different. Moreover the initial model has irreducible terms using selectors such as $car(nil)$. Because those terms correspond to execution exceptions e.g. unexpected segmentation violation, it is not suitable to use an initial model of a TRS when we use TRSs as functional programs.

The meaning that computation results are proper corresponds to normal forms are constructed by only constructors. The model of TRSs should exclude any term whose normal term has selectors and any term whose arbitrary subterm derives normal term with selectors.

Therefore we introduce a new concept about domain and model on TRSs. A constructive domain is a set of ground terms whose every subterm has a reduction sequence toward a ground constructor term on the domain. A constructive model is a set of congruent closures on the constructive domain of the TRS.

Definition 4.1 *Let $\langle R, \langle C, S, D\rangle, -\rangle$ is a TRS where $F = C \cup S \cup D$, a constructive domain of the TRS $T_C^R(F)$ is defined as follows.*

$$\tau \in T_C^R(F) \stackrel{\text{def}}{=} \left\{\begin{array}{l} \tau \in T(C) \\ \tau \underset{R}{\rightarrow} \sigma \text{ where } \left\{\begin{array}{l} \tau \equiv f(\tau_1, \ldots, \tau_n) \\ f \in F, arity(f) = n \\ \{\tau_1, \ldots, \tau_n, \sigma\} \subseteq T_C^R(F) \end{array}\right. \end{array}\right.$$

Definition 4.2 *A constructive model $T_C^R(F)/\underset{R}{\sim}$ is quotient of $T_C^R(F)$ by $\underset{R}{\sim}$.*

Notation 4.3 *We use $\underset{R}{\approx}$ instead of $\underset{R}{\sim}$ to distinguish the case on $T_C^R(F)$ from the case on $T(F)$, and use just $\approx$ when R is obvious.*

Definition 4.4 *We say $M \sqsubseteq M'$ where M, M' are constructive models if the following holds.*

$$M \models \tau = \sigma \Rightarrow M' \models \tau = \sigma$$

Lemma 4.5 *For every term in a constructive domain, there exists a reduction sequence from it toward a ground constructor term.*

General speaking $T(C) \subseteq T_C^R(F) \subseteq T(F)$ holds. Note that a constructive domain of a TRS is a subset of the restricted domain whose terms just have equivalent ground constructor terms. For example, the latter domain has a term $car(cons(nil, car(nil)))$ but the constructive domain does not.

5 Effects by transformation

We investigate properties of the transformation method in this section. First we prove that the method preserve models. Second we prove the method holds strong terminating property of TRSs. Finally we show that the method will break weak termination and confluency.

5.1 Models

Definition 5.1 *Where there exist R and p such that $P = R \cup \{p\}$ and $Q \supseteq R$, we say $\langle Q, F, - \rangle$ simulates $\langle P, F, - \rangle$ if the following holds.*

$$\forall \{\tau_1, \ldots, \tau_n, \sigma\} \subseteq T_C^P(F), \forall \tau (\equiv f(\tau_1, \ldots, \tau_n), f \in F)\ (\{\tau_1, \ldots, \tau_n, \sigma\} \subseteq T_C^Q(F) \wedge \tau \xrightarrow[P]{p} \sigma) \Rightarrow \tau \underset{Q}{\approx} \sigma$$

Lemma 5.2 *if $\langle P, F, - \rangle$ is simulated by $\langle Q, F, - \rangle$ then $T_C^P(F)/\underset{P}{\approx} \sqsubseteq T_C^Q(F)/\underset{Q}{\approx}$.*

Proof We show that a term in $T_C^P(F)$ is also in $T_C^Q(F)$ and the congruency on $T_C^P(F)$ holds on $T_C^Q(F)$ by induction on the structure of the term in $T_C^P(F)$.
base case: $\tau \in T_C^P(F) \Rightarrow \tau \in T_C^Q(F)$ holds obviously.
induction step: We assume the property for $\{\tau_1, \ldots, \tau_n, \sigma\} \subseteq T_C^P(F)$. Let τ be $f(\tau_1, \ldots, \tau_n)$.

1. Case $\tau \xleftrightarrow[P]{r} \sigma, r \neq p$: As r is common for P and Q, $\tau \xrightarrow[Q]{r} \sigma$ that is $\tau \underset{Q}{\approx} \sigma$.

2. Case $\tau \xrightarrow[P]{p} \sigma$: $\{\tau_1, \ldots, \tau_n, \sigma\} \subseteq T_C^Q(F)$ holds from the hypothesis of the induction. Here we have $\tau \underset{Q}{\approx} \sigma$ because $\langle Q, F, - \rangle$ simulates $\langle P, F - \rangle$.

Therefore the property holds for the all terms of $T_C^Q(F)$. ∎

Theorem 5.3 *The constructive models of a TRS and a transformed one by the method are the same.*

Proof (Brief) $\langle P, F', - \rangle$ and $\langle Q, F', - \rangle$ simulate each other on the transformationI so $T_C^{P'}(F')$ and $T_C^Q(F')$ are isomorphic by lemma5.2. $\langle P, F, - \rangle$ and $\langle Q, F, - \rangle$ simulate each other on the transformationII and on the transformationIII so $T_C^P(F)$ and $T_C^Q(F)$ are isomorphic by lemma5.2. ∎

5.2 Strong termination

Theorem 5.4 *A transformed TRS is strongly terminating if the original TRS is so.*

Proof (Outline) If we assume a transformed TRS is not strongly terminating then we can make a infinite reduction sequence in the original TRS. This is a contraposition of the theorem. ∎

5.3 Weak termination

Theorem 5.5 *A transformed TRS may not be weakly terminating even if the original TRS is so.*

Proof (Brief) The following is a contradiction. $\langle Q, \langle C, S, D\rangle, -\rangle$ is transformed from $\langle P, \langle C, S, D\rangle, -\rangle$ where $C = \{c_1, c_0\}, S = \{s_1^1\}, D = \{a, b, d\}$.

$$P: \left\{ \begin{array}{rcl} a(x) & \triangleright & b(s_1^1(x)) \\ b(x) & \triangleright & c_0 \\ d(x) & \triangleright & a(d(x)) \\ s_1^1(c_1(x)) & \triangleright & x \end{array} \right\} \quad Q: \left\{ \begin{array}{rcl} a(x) & \triangleright & h(x) \\ h(c_1(x)) & \triangleright & b(x) \\ b(x) & \triangleright & c_0 \\ d(x) & \triangleright & a(d(x)) \\ s_1^1(c_1(x)) & \triangleright & x \end{array} \right\}$$

$\langle P, \langle C, S, D\rangle, -\rangle$ is weakly terminating but $\langle Q, \langle C, S, D\rangle, -\rangle$ is not. ∎

5.4 Confluency

Theorem 5.6 *A transformed TRS may not be confluent even if the original TRS is so.*

Proof (Brief) The following is a contradiction. $\langle Q, \langle C, S, D\rangle, -\rangle$ is transformed from $\langle P, \langle C, S, D\rangle, -\rangle$ where $C = \{c_0, c_1\}, S = \{s_1^1\}, D = \{a, b\}$.

$$P: \left\{ \begin{array}{rcl} a(x, c_0) & \triangleright & x \\ a(x, c_1(y)) & \triangleright & a(s_1^1(x), y) \\ b & \triangleright & a(s_1^1(c_0), c_0) \\ b & \triangleright & a(c_0, c_1(c_0)) \end{array} \right\} \quad Q_1: \left\{ \begin{array}{rcl} a(x, 0) & \triangleright & x \\ a(x, c_1(y)) & \triangleright & a(s_1^1(x), y) \\ b & \triangleright & h(c_0) \\ h(c_1(x)) & \triangleright & a(x, 0) \\ b & \triangleright & a(c_0, c_1(c_0)) \end{array} \right\}$$

$\langle P, \langle C, S, D\rangle, -\rangle$ is confluent because it is left linear, right linear and strongly closed, but $\langle Q_1, \langle C, S, D\rangle, -\rangle$ is not confluent. ∎

6 Sufficient conditions for stable TRSs

We showed examples that the transformation method loosed weak termination and confluency in the section5. In this section, we provide some sufficient conditions for TRSs whose weak termination and confluency are not broken by the method.

6.1 Constructive TRSs

We have discussed about arbitrary TRSs. A constructive domain and a constructive model of any TRS can be defined if we can separate functions into three groups. This means that we introduced only the models that have a functional programming sense.

Now we deal with restricted TRSs, named constructive TRSs, that enjoy a manner of functional programming.

Definition 6.1 *we say a TRS $\langle R, \langle C, S, D\rangle, -\rangle$ is constructive if for any $\tau \underset{R}{\rightarrow} \sigma$ both $\tau \in T_C^R(F) \Rightarrow \sigma \in T_C^R(F)$ and $\tau \in T(C) \Rightarrow \sigma \in T(C)$ hold.*

Definition 6.2 *A constructive kernel of $\langle R, \langle C, S, D\rangle, -\rangle$ is a sub-TRS whose rewriting set are $\tau \rightarrow \sigma$ where $\tau, \sigma \in T(C)$.*

Lemma 6.3 *A constructive TRS is weakly terminating if and only if the constructive kernel of the TRS is weakly terminating*

Proof (Exercise) ∎

Theorem 6.4 *The transformation method preserves weakly terminating in a constructive domain of a constructive TRS.*

Proof (Brief) The transformation method does not affect a constructive kernel of the constructive TRS, so preserves weakly terminating. ∎

6.2 Regular TRSs

Regular TRSs are confluent[Hue80], but the transformation method can break regularity of a TRS. We provide a syntactical condition called strong regularity, which is a little bit more constrained than regularity. The transformation method preserves strong regularity of TRSs.

Definition 6.5 *A TRS is strongly regular if the TRS is regular and it does not have a rule such that* $c_n(\tau_1, \ldots, \tau_n) \triangleright \sigma$ *where* c_n *is a constructor and* $\tau_1, \ldots, \tau_n, \sigma$ *are arbitrary terms.*

Theorem 6.6 *A transformed TRS is strongly regular if the original TRS is strongly regular.*

Proof We show in the case of transformation I. In the case of transformation II is similar, and they are obvious in the other cases.
Let $\langle P = R \cup \{p\}, F, - \rangle$ is an original TRS and $\langle Q = R \cup \{q_1, q_2\}, F \cup \{h\}, - \rangle$ is an transformed one where p, q_1, q_2 are the following.

$$p\colon \gamma \triangleright \delta[s_n^{m_1}(\tau), \ldots, s_n^{m_k}(\tau), \tau, \ldots]$$
$$q_1\colon \gamma \triangleright h(\tau, y_1, \ldots, y_l)$$
$$q_2\colon h(c_n(x_1, \ldots, x_n), y_1, \ldots, y_l) \triangleright \delta[x_{m_1}, \ldots, x_{m_k}, c_n(x_1, \ldots, x_n), \ldots]$$

linearity: The lhs of q_1 is linear because the lhs of p is linear and the lhs of q_2 is linear by its syntax.
overlapping: The all lhs of R does not overlap γ that is the lhs of p because the original TRS is strongly regular. So they do not overlap the lhs of q_1. The lhs of q_2 does not overlap because the function symbol h does not appear in any rule of P and any lhs of $P \cup \{q_1\}$ is not a term whose main function is c_n.
Therefore Q is strongly regular. ∎

7 Conclusion

We had observed a role of functions in the TRS and introduced a classification of them. The transformation method provided in this paper eliminates selectors of a TRS. The method does not effect to constructive model while transformation, and preserves strong termination. Unfortunately the method can break weak termination and confluency. We provided two sufficient conditions of TRSs; the first preserves weak termination in the constructive domains and the other preserves confluency. The table1 shows the properties of change in the transforming TRSs.

We must solve the following in the next step.

- Define conservative extended models of constructive models. The discussion about preserving models will be clear on conservative extended models when using the transformation I.
- Theorem 6.4 will be a meaningful if the transformation preserves constructive property for an arbitrary constructive TRS.

- The transformation method preserves confluency in a constructive domain of a constructive TRS.
- Relax the definition of selectors. A function *2nd*in the example 3.1 is a defined operator here but it may be interpreted as a selector.

TRS	no constraint				constructive		strongly regular	
property	Model	ST	WT	CR	WT	CR	WT	CR
TransI	△	○	×	×	○	−	−	○
TransII	◎	○	×	×	○	−	−	○
TransIII	◎	○	○	−	○	−	−	○
TransIV	△	○	−	−	○	−	−	○
The whole method	△	○	×	×	○	−	−	○

ST: strong termination WT: weak termination CR: confluency

◎ the same model
△ the same extended model (not the same model)

○ preserves
× does not preserve
− unknown

Table 1: Properties of transformed TRSs

References

[BK86] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Noth-Holland, Amsterdam, 1990.

[GM87] E. Giovannetti and C. Moiso. Notes on the elimination of conditions. In S.Kaplan and J.-P.Jouannaud, editors, *Lecture Notes in Computer Science, 308*, pages 91–97. Springer-Verlag, 7 1987.

[HH82] G. Huet and J. M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer System Science*, 25(2):239–266, 1982.

[Hue80] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[Kik95] Yutaka Kikuchi. Eliminating selectors from term rewriting systems. Technical Report 95TR-0014, Tokyo Institute of Technology, Ookayama, Meguro, Tokyo 152 Japan, 7 1995. ISSN 0918-2802, in Japanese.